# Fiche TD7 : Algorithmique des graphes

**Exercice 1 (*)** *Couplages dans un graphe quelconque*

1. Donner une condition nécessaire pour qu'un graphe connexe admette un couplage parfait. Est-elle suffisante ?
2. Montrer que pour $n \geq 2$, il existe un graphe connexe à $n$ sommets pour lequel la cardinalité maximale d'un couplage vaut un.
3. Si $C$ et $C'$ sont deux couplages dans un même graphe et que $C$ est maximal, montrer que $|C'| \leq 2|C|$.

**Exercice 2 (**)** *Détection de graphes bipartis*

On considère l'algorithme suivant, prenant en entrée un graphe non orienté connexe $G$ et un sommet $s_0$ de $G$ :

```
mystère(G, s_0) =
    Poser numéro(s_0) = 1
    Mettre (s_0, 1) dans un sac
    Tant que le sac n'est pas vide
        Sortir un couple (s, c) du sac
        Pour tout voisin t de s
            Si numéro(t) = c
                Renvoyer faux
            Sinon, si t n'a pas encore de numéro
                Poser numéro(t) = 3 − c
                Mettre (t, 3 − c) dans le sac
    Renvoyer vrai
```

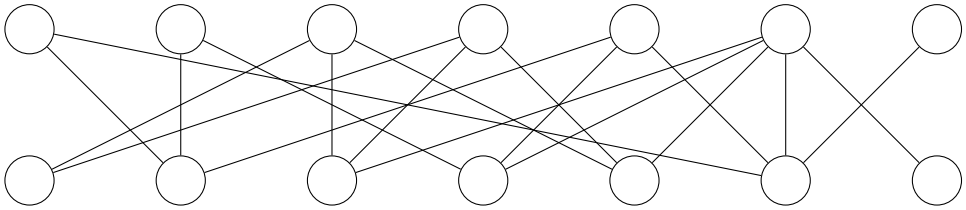
1. Donner une spécification précise pour cet algorithme et justifier sa correction vis-à-vis de cette spécification.
2. En précisant la façon dont seraient implémentées les structures de données, déterminer sa complexité.
3. Comment ferait-on pour adapter cet algorithme aux graphes non connexes ?

*Remarque : Cet algorithme repose sur le fait qu'il est équivalent pour un graphe d'être biparti ou d'être 2-coloriable. On dispose au passage d'un algorithme de test polynomial de 2-coloriabilité, tandis que la $k$-coloriabilité pour $k > 2$ est un problème NP-complet.*

**Exercice 3 (***)** *Théorème de König*

Si $G$ est un graphe non orienté, non pondéré, on appelle *transversal* de $G$ un sous ensemble $T$ des sommets de $G$ telle que toute arête de $G$ est incidente à au moins un sommet de $T$ ($T$ "couvre" les arêtes). On dit qu'un transversal de $G$ est minimum si aucun transversal de $G$ n'a un cardinal strictement plus petit.

1. Si $M$ est un couplage de $G$ et $T$ un transversal de $G$, montrer que $|M| \leq |T|$.
2. Le résultat précédent montre en particulier que la taille d'un couplage maximum dans un graphe quelconque est inférieure à la taille d'un transversal minimum. A-t-on égalité entre ces deux quantités en général ?
3. On suppose à présent que $G$ est un graphe biparti dont on note $U \sqcup V$ la bipartition des sommets et $M$ un couplage maximum. On note par ailleurs $E$ l'ensemble formé des sommets de $U$ non saturés par $M$ auxquels on ajoute les sommets atteignables à partir d'un de ces sommets par un chemin $M$-alternant.
    a) Montrer que pour toute arête $(u, v)$ de $M$ on a soit $u, v \in E$ soit $u, v \notin E$.
    b) Montrer que $T = (U \setminus E) \cup (V \cap E)$ est de cardinal égal à $|M|$.
    c) Montrer que $T$ est un transversal de $G$.
    d) Déduire de ce qui précède le théorème de König : dans un graphe biparti, la taille d'un couplage maximum est égale à la taille d'un transversal minimum.
4. Déterminer un transversal minimum dans le graphe suivant :



*Remarques : Le problème du calcul d'un transversal minimum est un problème connu sous le nom de Vertex Cover et est NP-complet dans un graphe quelconque. Le théorème de König, tout comme le théorème de Hall vu l'année dernière, est un cas particulier du théorème flot-max / coupe-min, qui est un résultat important en optimisation.*

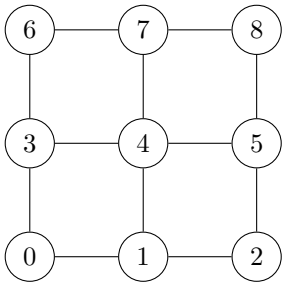
**Exercice 4 (**)** *Whatever First Search*

On considère deux versions de l'algorithme générique de parcours vu en cours (la version de gauche est exactement celle du cours) auquel on ajoute de l'information pour reconstruire les arbres de parcours :

```
WFS1(G, s_0) =
    Mettre (−1, s_0) dans le sac
    Tant que le sac n'est pas vide
        Sortir un élément (p, s) du sac
        Si s n'est pas marqué
            Marquer s
            Ajouter (p, s) à l'arbre de parcours
            Pour tout voisin t de s
                Mettre (s, t) dans le sac
```

```
WFS2(G,s_0) =
    Mettre (−1, s_0) dans le sac
    Marquer s_0
    Tant que le sac n'est pas vide
        Sortir un élément (p, s) du sac
        Ajouter (p, s) à l'arbre de parcours
        Pour tout voisin t de s
            Si t n'est pas marqué
                Mettre (s, t) dans le sac
                Marquer t
```

1. On suppose que le sac est une file. Déterminer si les données suivantes changent entre les deux parcours :
   a) L'ordre de parcours.
   b) La complexité temporelle.
   c) La complexité spatiale.
2. On suppose que le sac est une pile. En supposant que lorsqu'un choix est possible on traite les sommets par ordre croissant de numéros, appliquer WFS1 au graphe suivant et déterminer l'ordre dans lequel les sommets sont traités et l'arbre du parcours. Faire de même pour WFS2 et comparer.



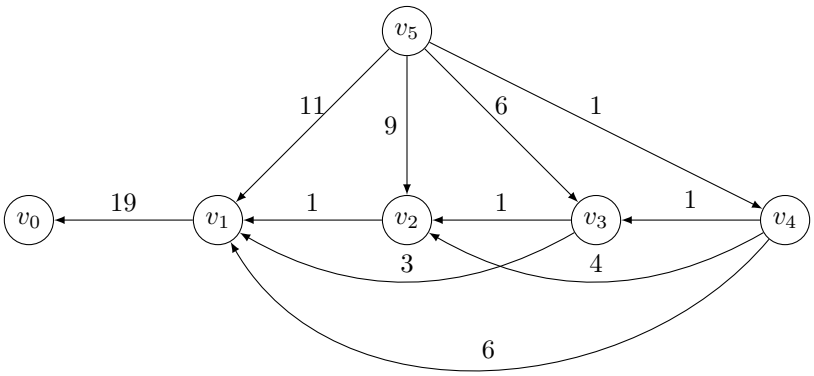

**Exercice 5 (*)** *Parcours en profondeur précis*

On considère la version du parcours en profondeur donnée en cours permettant la classification des arcs. Les tableaux de début et fin d'exploration d'un sommet sont notés $D$ et $F$. Dans chacun des cas suivants, dire en justifiant si l'affirmation est correcte :

1. Si $y$ est accessible depuis $x$ alors on initialisera $D[y]$ pendant l'exploration de $x$.
2. Si $y$ est accessible depuis $x$ et $D[x] < D[y]$ alors $D[y] < F[x]$.
3. S'il existe un arc $(x, y)$ et que $D[x] < D[y]$ alors $D[y] < F[x]$.

**Exercice 6 (*)** *Heuristiques pour $A^*$*

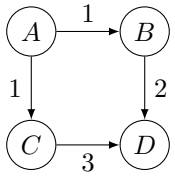
On souhaite déterminer un plus court chemin entre la source $v_5$ et la destination $v_0$ à l'aide de l'algorithme $A^*$. Les valeurs de l'heuristique $h$ pour chaque sommet sont indiquées ci-dessous :




| Sommet $v$ | $v_0$ | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ |
|---|---|---|---|---|---|---|
| Valeur de $h(v)$ | 0 | 0 | 3 | 7 | 13 | 23 |

1. L'heuristique $h$ est-elle admissible ? Est-elle monotone ?
2. Appliquer l'algorithme de Dijkstra à cet exemple puis lui appliquer $A^*$. Commenter.

On considère à présent le graphe $G$ suivant :




3. Proposer une heuristique monotone pour $G$.
4. Proposer une heuristique admissible mais non monotone pour $G$.
5. Proposer une heuristique non admissible mais telle que le déroulé de $A^*$ renvoit le résultat escompté lorsqu'on tente de calculer un plus court chemin de $A$ à $D$.
6. Proposer une heuristique pour laquelle $A^*$ ne renvoit pas un plus court chemin de $A$ à $D$.

**Exercice 7 (*)** *Kosaraju et satisfiabilité*

En construisant le graphe associé à chacune des formules suivantes et à l'aide de l'algorithme de Kosaraju, déterminer si les formules suivantes sont satisfiables et en donner un modèle si c'est le cas.

1. $\psi = (\neg t \vee \neg z) \wedge (z \vee \neg y) \wedge (x \vee t) \wedge (z \vee x) \wedge (y \vee t)$.
2. $\varphi = (x \vee z) \wedge (y \vee x) \wedge (\neg z \vee \neg y) \wedge (y \vee z) \wedge \neg x$.

**Exercice 8 (**)** *Arbres couvrants minimaux et arêtes de poids minimal*

On considère un graphe $G = (S, A)$ non orienté, pondéré via une fonction de pondération $p$ injective (tous les poids des arêtes sont donc distincts) et tel que $|A| \geq 3$. Soit $T$ un arbre couvrant minimal de $G$.

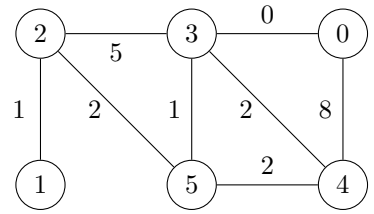
1. Si $T'$ est un arbre couvrant minimal de $G$, montrer que $T = T'$.
2. Montrer que $T$ contient les deux arêtes de poids minimal de $A$.
3. Montrer que $T$ ne contient pas nécessairement les trois arêtes de poids minimal de $A$.
4. Montrer que pour chaque cycle de $G$, $T$ ne contient pas l'arête de poids maximal du cycle.
5. $T$ contient-il l'arête de poids minimal de chaque cycle ?

**Exercice 9 (**)** *Algorithme de Prim*

On considère un graphe $G = (S, A)$ non orienté, connexe, pondéré par une fonction de pondération $p$. On souhaite calculer un arbre couvrant minimal pour $G$ via l'algorithme de Prim-Jarník décrit ci-après :

```
Prim(G) =
    E ← ∅
    V ← {s_0} où s_0 est un élément de S pris au hasard
    F ← S \ {s_0}
    Tant que F ≠ ∅
        Choisir une arête (u, v) ∈ A telle que u ∈ V, v ∉ V et p(u, v) est minimal
        F ← F \ {v}
        E ← E ∪ {(u, v)}
        V ← V ∪ {v}
    Renvoyer (V, E)
```

1. Appliquer cet algorithme au graphe suivant avec $s_0 = 0$.




2. Montrer qu'à tout instant, $V$ et $F$ forment une partition de l'ensemble $S$ des sommets.

3. Montrer la terminaison de cet algorithme.

4. Montrer que "le graphe $(V, E)$ est inclus dans un arbre couvrant minimal de $G$" est un invariant pour la boucle tant que de cet algorithme. En déduire la correction de l'algorithme.

Pour implémenter cet algorithme, on numérote consécutivement à partir de 0 les sommets de $G$. On associe à chaque sommet $s$ une clé correspondant à la distance minimale (en une arête) de $s$ à l'ensemble $V$. On implémente alors $F$ par une file de priorité min vis à vis de ces clés et on maintient en parallèle un tableau $P$ indiquant à la case $s$ quel est le sommet de $V$ le plus proche de $s$ (valant $-1$ si aucun voisin du sommet n'est dans $V$).

5. Détailler le contenu de $F$ et $P$ au fil de l'exécution de l'algorithme sur le graphe de la question 1.

6. En supposant que la file de priorité $F$ est implémenté par un tas min, quelle est la complexité de l'algorithme de Prim ?

**Exercice 10 (***)** *Complexités pour union-find*

Dans cet exercice on cherche à estimer la complexité amortie d'une opération trouver ou unir dans une structure union-find avec union par rang et compression de chemin. Le logarithme itéré, note $\log^*$, est défini pour tout $x \in \mathbb{R}$ par :

$$\log^* = \begin{cases} 0 & \text{si } x \leq 1 \\ 1 + \log^*(\log_2(x)) & \text{si } x > 1 \end{cases}$$

1. Quelle est la plus grande valeur de $x$ pour laquelle $\log^*(x) \leq 4$ ? Et pour laquelle $\log^*(x) \leq 5$ ? Pour comparaison, le nombre d'atomes dans l'univers est de l'ordre de $2^{80}$.

2. On définit ${}^k x = x^{x^{\cdot^{\cdot^{x}}}}$ avec $k$ étages d'exponentiation et la convention ${}^0 x = 1$. Montrer que pour tout $k \in \mathbb{N}^\star$, $\log^*(x) = k$ si et seulement si $x \in ]{}^{k-1}2, {}^k 2]$.

On définit à présent $\varphi(k) = {}^k 2$ et on regroupe les noeuds d'un arbre représentant une partie dans union-find par niveau : le niveau $k$ contient tous les noeuds dont le rang (pas la hauteur !) est dans l'intervalle $[\varphi(k), \varphi(k+1) - 1]$. Le nombre total d'éléments dans la partition est noté $n$.

3. Montrer que le nombre de niveaux non vides est majoré par $\log^*(n)$.

4. Montrer qu'un noeud au rang $r$ possède au moins $2^r$ descendants.

5. Montrer qu'il y a au plus $2n/\varphi(k+1)$ noeuds dans le niveau $k$.

6. Montrer que le rang croît strictement le long de tout chemin reliant un noeud à la racine de son arbre.

7. Montrer que si un noeud change de parent, son nouveau père a un rang strictement supérieur à celui de l'ancien père.

8. Montrer que le coût total d'une série de $m$ opérations pouvant être soit trouver soit unir est en $O(m + C)$ où $C$ est le nombre de changements de parents lors de ces opérations.

9. En déduire que le coût total d'une série de $m$ opérations est en $O(m \log^*(n))$.

*Remarque : Cette dernière question montre que la complexité amortié d'une opération trouver ou unir dans ce contexte en en* $O(\log^*(n))$. *Cette complexité est en fait une majoration grossière : on peut montrer qu'elle est en* $O(\alpha(n))$ *où* $\alpha$ *croît encore plus lentement que* $\log^*$. *Mais comme* $\log^*(n)$ *ne dépasse déjà* 5 *que pour des valeurs incroyablement grandes de* $n$, *cela suffit à se convaincre que la complexité d'un unir ou trouver avec union par rang et compression de chemin est "essentiellement constante".*